ASSIGNMENT NO.6

Title: Implement the Continuous Bag of Words (CBOW) Model.

Aim: Implement the Continuous Bag of Words (CBOW) Model. Stages can be:
Data preparation
Generate training data
Train model
Output

Theory:

NLP stands for Natural Language Processing. It is a field of artificial intelligence that focuses on the interaction between computers and human language. NLP aims to enable computers to understand, interpret, and generate human language in a valuable way. This field involves various tasks, such as text classification, sentiment analysis, language translation, speech recognition, and text generation.

NLP is growing increasingly sophisticated, yet much work remains to be done. Current systems are prone to bias and incoherence, and occasionally behave erratically. Despite the challenges, machine learning engineers have many opportunities to apply NLP in ways that are ever more central to a functioning society.

Word embedding in NLP refers to the technique of representing words as continuous vector spaces in a way that captures their semantic relationships and contextual meanings. Instead of using traditional one-hot encoding, where words are represented as binary vectors, word embedding assigns each word a vector in a continuous space. This allows NLP models to learn the meaning and relationships between words based on their context and usage.

Word2Vec is a machine learning technique that has been around since 2013, courtesy of Tomas Mikolov and his data science team at Google. It relies on deep learning to train a computer to learn about your language (vocabulary, expressions, context, etc.) using a corpus (content library).

Word2Vec is a popular word embedding technique in NLP developed by Google. It aims to learn word embeddings by training a neural network on a large corpus of text data. There are two main approaches with Word2Vec:

a) Continuous Bag of Words (CBOW): This approach predicts a target word based on its context, i.e., the words surrounding it in a sentence.

b) Skip-gram: In this approach, the model predicts the context words based on a target word.

Applications of word embeddings in NLP include: a)

Document and text classification

b) Sentiment analysis

c) Machine translation

d) Named entity recognition

e) Information retrieval

f) Text summarization

g) Speech recognition

h) Question answering systems

i) Recommendation systems

The CBOW model tries to understand the context of the words and takes this as input. It then tries to predict words that are contextually accurate. Let us consider an example for understanding this. Consider the sentence: 'It is a pleasant day' and the word 'pleasant' goes as input to the neural network. We are trying to predict the word 'day' here. We will use the one-hot encoding for the input words and measure the error rates with the one-hot encoded target word. Doing this will help us predict the output based on the word with least error.

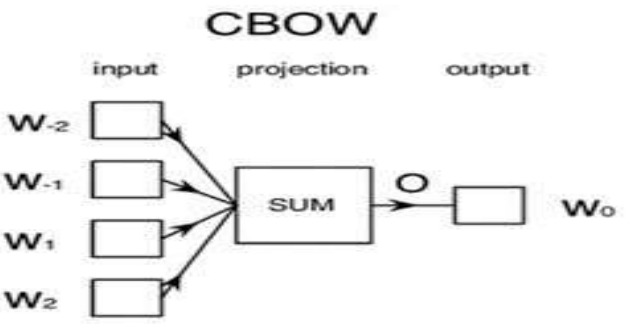

Figure 1: CBOW model

The CBOW model architecture is as shown above. The model tries to predict the target word by trying to understand the context of the surrounding words. Consider the same sentence as above, 'It is a pleasant day'.The model converts this sentence into word pairs in the form

(contextword, targetword). The user will have to set the window size. If the window for the context word is 2 then the word pairs would look like this: ([it, a], is), ([is, pleasant], a),([a, day], pleasant). With these word pairs, the model tries to predict the target word considered the context words.

If we have 4 context words used for predicting one target word the input layer will be in the form of four 1XW input vectors. These input vectors will be passed to the hidden layer where it is multiplied by a WXN matrix. Finally, the 1XN output from the hidden layer enters the sum layer where an element-wise summation is performed on the vectors before a final activation is performed and the output is obtained.

Input to CBOW model and Output to CBW model.

It is trained using a feedforward neural network where the input is a set of context words, and the output is the target word. The model learns to adjust the weights of the neurons in the hidden layer to produce an output that is most likely to be the target word.

Tokenization is the first step in any NLP pipeline. It has an important effect on the rest of your pipeline. A tokenizer breaks unstructured data and natural language text into chunks of information that can be considered as discrete elements. The token occurrences in a document can be used directly as a vector representing that document.

This immediately turns an unstructured string (text document) into a numerical data structure suitable for machine learning. They can also be used directly by a computer to trigger useful actions and responses. Or they might be used in a machine learning pipeline as features that trigger more complex decisions or behaviour.

Tokenization can separate sentences, words, characters, or subwords. When we split the text into sentences, we call it sentence tokenization. For words, we call it word tokenization. Example of sentence tokenization

```
sent_tokenize('Life is a matter of choices, and every choice you make makes you.')
['Life is a matter of choices, and every choice you make makes you.']
```

The window size parameter in the CBOW model specifies the number of context words to consider on either side of the target word. For example, if the window size is set to 2, the model will consider two words to the left and two words to the right of the target word as the context. The window size parameter influences the amount of contextual information the model considers during training.

In deep learning, **embedding layers** plays a vital role, especially when dealing with problems of NLP(Natural Language Processing). The embedding layer requires integer encoded input data to represent each word uniquely. To prepare data, we can use TokenizerAPI provided by Keras.

The embedding layers are flexible because they are used in transfer learning to load a pretrained word embedding model. Also, embedding layers can be part of the deep learning model where embedding is done along with the model itself.

The first hidden layer of the network is the embedding layer. The output of the embedding layer is a 2D vector where every vector represents a single word from the vocabulary.

The Lambda layer exists so that arbitrary expressions can be used as a Layer when constructing Sequential and Functional API models. Lambda layers are best suited for simple operations or quick experimentation. For more advanced use cases, follow this guide for subclassing tf.keras.layers.Layer.

The main reason to subclass tf.keras.layers.Layer instead of using a Lambda layer is saving and inspecting a Model. Lambda layers are saved by serializing the Python bytecode, which is fundamentally non-portable. They should only be loaded in the same environment where they were saved. Subclassed layers can be saved in a more portable way by overriding their get_config() method. Models that rely on subclassed Layers are also often easier to visualize and reason about.

yield () is a Python keyword used in generator functions. When you use yield in a function, it turns that function into a generator. Instead of returning a value, a generator yields a value one at a time. It allows you to iterate over a potentially large sequence of values without storing them all in memory at once, which can be memory-efficient. Generator functions are often used for tasks like processing large datasets or generating sequences of values on the fly.

Steps/ Algorithm

Dataset link and libraries :

Create any English 5 to 10 sentence paragraph as input Import following data from keras :

keras.models import Sequential keras.layers import Dense, Embedding, Lambda

keras.utils import np_utils keras.preprocessing

import sequence keras.preprocessing.text import

Tokenizer Import Gensim for NLP operations :

requirements :

Gensim runs on Linux, Windows and Mac OS X, and should run on any other platform that supports Python 3.6+ and NumPy. Gensim depends on the following software: Python, tested with versions 3.6, 3.7 and 3.8. NumPy for number crunching.

Ref: https://analyticsindiamag.com/the-continuous-bag-of-words-cbow-model-in-nlp-handson- implementation-with-codes/

Import following libraries gemsim and numpy set i.e. text file created . It should be preprocessed.

Tokenize the every word from the paragraph . You can call in built tokenizer present in Gensim

Fit the data to tokenizer

Find total no of words and total no of sentences. Generate

the pairs of Context words and target words :

e.g. cbow_model(data, window_size, total_vocab): total_length = window_size*2

for text in data: text_len = len(text) for idx, word in enumerate(text):

context_word = [] target = []

begin = idx - window_size end = idx + window_size + 1

context_word.append([text[i] for i in range(begin, end) if 0 <= i < text_len and i

!= idx]) target.append(word) contextual = sequence.pad_sequences(context_word,

total_length=total_length) final_target = np_utils.to_categorical(target, total_vocab)

yield(contextual, final_target)

Create Neural Network model with following parameters . Model type : sequential

Layers : Dense , Lambda , embedding. Compile Options : (loss='categorical_crossentropy', optimizer='adam')

Create vector file of some word for testing e.g.:dimensions=100 vect_file =

open('/content/gdrive/My Drive/vectors.txt' ,'w') vect_file.write('{}

{}\n'.format(total_vocab,dimensions) Assign

weights to your trained model

e.g. weights = model.get_weights()[0] for text, i in vectorize.word_index.items():

```
final_vec = ' '.join(map(str, list(weights[i, :]))) vect_file.write('{} {}\n'.format(text, final_vec)
Close()
```

Use the vectors created in Gemsim :

e.g. cbow_output = gensim.models.KeyedVectors.load_word2vec_format('/content/gdrive/My Drive/vectors.txt', binary=False) choose the word to get similar type of words:

```
cbow_output.most_similar(positive=['Your word'])
```

INPUT CODE:

```
import matplotlib.pyplot as plt import seaborn as sns import matplotlib as mpl import
matplotlib.pylab as pylab import numpy as np %matplotlib inline import re


sentences = """We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules called a
program. People create programs to direct processes. In effect, we
conjure the spirits of the computer with our spells."""

# remove special characters sentences =
re.sub('[^A-Za-z0-9]+', ' ', sentences)
# remove 1 letter words sentences = re.sub(r'(?:^| )\w(?:$|
)', ' ', sentences).strip()
# lower all characters sentences
= sentences.lower()
#Vocabulary  words =
sentences.split() vocab =
set(words) vocab_size =
len(vocab) embed_dim
= 10 context_size = 2

#Implementation word_to_ix = {word: i for i, word in
enumerate(vocab)} ix_to_word = {i: word for i, word in
enumerate(vocab)} #Data Bags
# data - [(context), target] data
= [] for i in range(2, len(words)
- 2):
   context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
target = words[i]     data.append((context, target)) print(data[:5])

#Embeedings  embeddings =  np.random.random_sample((vocab_size,
embed_dim))
```

```
#Linear model def
linear(m, theta):
w = theta     return
m.dot(w)
#Log softmax + NLLloss = Cross Entropy def
log_softmax(x):
   e_x   =   np.exp(x   -   np.max(x))
return np.log(e_x / e_x.sum())

def NLLLoss(logs, targets):     out =
logs[range(len(targets)), targets]
return -out.sum()/len(out)

def log_softmax_crossentropy_with_logits(logits,target):
   out = np.zeros_like(logits)     out[np.arange(len(logits)),target] = 1
softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True)
return (- out + softmax) / logits.shape[0]

#Forward        Function        def
forward(context_idxs, theta):
m = embeddings[context_idxs].reshape(1, -1)
n  = linear(m, theta)     o = log_softmax(n)     return m, n,o

#Backward            Function            def
backward(preds, theta, target_idxs):
   m, n, o = preds     dlog =
log_softmax_crossentropy_with_logits(n, target_idxs)     dw =
m.T.dot(dlog)     return dw

#Optimize        function        def
optimize(theta, grad, lr=0.03):
   theta   -=   grad   *   lr
return theta
```

```
#Training
#Genrate training data theta = np.random.uniform(-1, 1, (2 * context_size *
embed_dim, vocab_size)) epoch_losses = {}

for epoch in range(80):
   losses =  []     for context,
target in data:
      context_idxs = np.array([word_to_ix[w] for w in context])
preds = forward(context_idxs, theta)        target_idxs =
np.array([word_to_ix[target]])        loss = NLLLoss(preds[-1],
target_idxs)        losses.append(loss)

      grad = backward(preds, theta, target_idxs)
theta = optimize(theta, grad, lr=0.03)
epoch_losses[epoch] = losses

#Analyze
#Plot  loss/epoch  ix
= np.arange(0,80)

fig = plt.figure() fig.suptitle('Epoch/Losses',
fontsize=20) plt.plot(ix,[epoch_losses[i][0]
for i in ix]) plt.xlabel('Epochs',
fontsize=12) plt.ylabel('Losses',
fontsize=12)

#Predict function def
predict(words):
   context_idxs = np.array([word_to_ix[w] for w in words])
preds = forward(context_idxs, theta)     word =
ix_to_word[np.argmax(preds[-1])]     return word
```

```
# (['we', 'are', 'to', 'study'], 'about') predict(['we',
'are', 'to', 'study'])
#Accuracy  def accuracy():
wrong = 0     for context, target
in data:   if(predict(context) !=
target):            wrong += 1
return (1 - (wrong / len(data)))

accuracy() predict(['processes', 'manipulate',
'things', 'study'])
```

OUTPUT:

```
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib as mpl
import matplotlib.pylab as pylab
import numpy as np
%matplotlib inline
```

```
C:\Users\DELL\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumPy version >=1.16.5 a
ed for this version of SciPy (detected version 1.26.0
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

```
import re
```

```
sentences = """We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In effect,
we conjure the spirits of the computer with our spells."""
```

Figure 1: Output 1

The above figure shows the importing of libraries and dataset preparation.

```
#Log softmax + NLLloss = Cross Entropy

def log_softmax(x):
    e_x = np.exp(x - np.max(x))
    return np.log(e_x / e_x.sum())
```

```
def NLLLoss(logs, targets):
    out = logs[range(len(targets)), targets]
    return -out.sum()/len(out)
```

```
def log_softmax_crossentropy_with_logits(logits,target):

    out = np.zeros_like(logits)
    out[np.arange(len(logits)),target] = 1

    softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True)

    return (- out + softmax) / logits.shape[0]
```

Figure 2 : Output 2

The above figure shows the use of SoftMax function.

```python
#Forward Function

def forward(context_idxs, theta):
    m = embeddings[context_idxs].reshape(1, -1)
    n = linear(m, theta)
    o = log_softmax(n)

    return m, n, o
```

```python
#Backward Function

def backward(preds, theta, target_idxs):
    m, n, o = preds

    dlog = log_softmax_crossentropy_with_logits(n, target_idxs)
    dw = m.T.dot(dlog)

    return dw
```

Figure 3: Output 3

The above figure shows the use of Forward and Backward function.

```python
#Training

#Genrate training data

theta = np.random.uniform(-1, 1, (2 * context_size * embed_dim, vocab_size))
```

```python
epoch_losses = {}

for epoch in range(80):

    losses =  []

    for context, target in data:
        context_idxs = np.array([word_to_ix[w] for w in context])
        preds = forward(context_idxs, theta)

        target_idxs = np.array([word_to_ix[target]])
        loss = NLLLoss(preds[-1], target_idxs)

        losses.append(loss)

        grad = backward(preds, theta, target_idxs)
        theta = optimize(theta, grad, lr=0.03)
```

Figure 4: Output 4

The above figure shows the training of dataset.

```
In [25]:
# (['we', 'are', 'to', 'study'], 'about')
predict(['we', 'are', 'to', 'study'])

Out[25]: 'about'
```

```
In [26]:
#Accuracy

def accuracy():
    wrong = 0

    for context, target in data:
        if(predict(context) != target):
            wrong += 1

    return (1 - (wrong / len(data)))
```

```
In [27]:
accuracy()

Out[27]: 1.0
```

```
In [28]:
predict(['processes', 'manipulate', 'things', 'study'])

Out[28]: 'other'
```

Figure 5: Implemented model

The above figure shows the implementation of model with the accuracy 100%.

Conclusion:

Thus, the model is implemented and tested using deep learning function like SoftMax, Backward, Forward functions. The model is been tested and trained with the accuracy of 100%.